NPS-EC-92-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

MULTIPLE-VALUED PROGRAMMABLE LOGIC ARRAY MINIMIZATION BY SIMULATED ANNEALING

Gerard W. Dueck
Robert C. Earle
Parthasarathy Tirumalai
Jon T. Butler

February 10, 1992

Prepared for: Naval Research Laboratory
Washington, DC 20375

NAVAL POSTGRADUATE SCHOOL
Monterey, CA 93943

Rear Admiral R.W. West, Jr.
Superintendent

Professor H. Shull
Provost

This report was prepared for the **Naval Research Laboratory** and funded by the **Naval Postgraduate School.**

This report was prepared by: Prof. Gerhard W. Dueck, LT Robert C. Earle, Dr. Parthasarathy Tirumalai and Prof. Jon T. Butler

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS-EC-92-004	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) EC/Bu	7a NAME OF MONITORING ORGANIZATION Naval Research Laboratory
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5004	7b ADDRESS (City, State, and ZIP Code) Washington, DC 20375
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Postgraduate School 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O & MN, Direct Funding
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Multiple-Valued Programmable Logic Array Minimization by Simulated Annealing

12 PERSONAL AUTHOR(S)
Gerhard W. Dueck, Robert C. Earle, Parthasarathy Tirumalai and Jon T. Butler

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical Report	FROM 7/91 TO 2/92	10 February 1992	

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer-aided design tool, multiple-valued logic, programmable logic array, heuristic minimization technique, VLSI design tool.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

We propose a solution to the minimization problem of multiple-valued programmable logic arrays (PLA) that uses simulated annealing. The algorithm accepts a sum-of-products expression, divides and recombines the product terms, gradually progressing toward a minimal solution. The input expression can be user-specified or one produced by another heuristic. The process is termed simulated annealing because it has an analog in the statistical mechanical model of annealing in solids. That is, the slow cooling of certain solids results in a state of low energy, a crystalline state rather than an amorphous state that results from fast cooling. In a PLA, the crystalline state is analogous to a realization with a small number of product terms.

Unlike recently studied minimization techniques (which are classified as direct-cover methods), our technique manipulates product terms directly, breaking them up and joining them in different ways while reducing the total number of product terms.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Jon T. Butler	22b TELEPHONE (Include Area Code) 408-646-3299 / 22c. OFFICE SYMBOL EC/Bu

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

MULTIPLE-VALUED PROGRAMMABLE LOGIC ARRAY MINIMIZATION BY SIMULATED ANNEALING*

by

Gerhard W. Dueck†, Robert C. Earle†, Parthasarathy Tirumalai‡, and Jon T. Butler†

†Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5004

‡Hewlett-Packard Laboratories
Bldg. 3U-7, 1501 Page Mill Road
Palo Alto, CA 94304

October 22, 1991

* Research supported in part by the Naval Research Laboratory, Washington, DC through direct funds at the Naval Postgraduate School, Monterey, CA and by a National Research Council Research Associateship tenured at the Naval Postgraduate School, Monterey, CA.

ABSTRACT

We propose a solution to the minimization problem of multiple-valued programmable logic arrays (PLA) that uses simulated annealing. The algorithm accepts a sum-of-products expression, divides and recombines the product terms, gradually progressing toward a minimal solution. The input expression can be user-specified or one produced by another heuristic. The process is termed 'simulated annealing' because it has an analog in the statistical mechanical model of annealing in solids. That is, the slow cooling of certain solids results in a state of low energy, a crystalline state rather than an amorphous state that results from fast cooling. In a PLA, the crystalline state is analogous to a realization with a small number of product terms.

Unlike recently studied minimization techniques (which are classified as direct-cover methods), our technique manipulates product terms directly, breaking them up and joining them in different ways while reducing the total number of product terms. We show two mechanisms for recombining product terms and compare the results with presently known heuristics. Specifically, we compare both the number of product terms and the speed of execution. A unique feature of simulated annealing is that its execution time is controllable, allowing one to tradeoff time for minimality. It has been incorporated in the HAMLET PLA minimization tool.

I. INTRODUCTION

The only known algorithm for finding a minimal sum-of-products expression is exhaustive search. However, excessive computation time makes this approach impractical. For example, in a comparison of minimization algorithms on simple expressions [11], three days of computation time were required to produce minimal expressions by exhaustive search, while only three seconds were required for heuristic algorithms to produce *near* minimal expressions.

Sum-of-products expressions are interesting because of the ease with which they can be implemented by programmable logic arrays (PLA's). Implementation by PLA is easier than by random logic because the circuit designer needs only provide a design for the row-column intersection. This design appears throughout the PLA, where each occurrence is programmed by the user. Recent progress in the implementation of multiple-valued PLA's has occurred in CCD [5]. Implementations have been proposed for current-mode CMOS [15].

Because of the computational complexity associated with minimal sum-of-products solutions, there is considerable interest in heuristics. At least four are known; Pomper and Armstrong [7], Besslich [1], Dueck and Miller [3], and Yang and Wang [14]. All use the direct cover method; that is, first a minterm is selected and then an implicant is chosen that covers the minterm. This process is repeated until the given expression is covered. Using search in conjunction with the direct cover method [15], improves the realizations but increases the computation time. The increased computation time has inspired research into parallel minimization algorithms [12, 13].

We propose an alternative to the direct cover method. Instead of creating implicants, our algorithm manipulates existing implicants. That is, implicants are combined, reshaped, or divided. Manipulation of implicants is not new; it is used in binary minimization [2], and was proposed for multiple-valued sum-of-products expressions [8]. What is new is the means of manipulation. We do it nondeterministically. That is, randomly chosen implicants are randomly combined, reshaped, or divided. The number of implicants in a cover of an expression increases when an implicant is divided and during certain reshapings. However, this allows one to go from a false minimum to a true minimum. The algorithm is essentially a series of transitions from solution to solution in the solution space. As time goes on, the probability decreases that a divide or reshape that increases the number of product terms is performed. In so doing, the transitions among solutions become gradually biased towards solutions with fewer product terms.

The process suggested above is similar to the slow cooling of metals or glass, which allows the "melt" to reach a low energy state, a crystalline state. This is termed *annealing*, and the corresponding optimization method is called *simulated annealing*. *Slow* cooling is essential to the achievement of a minimal solution. On the contrary, in both the physical

system and the optimization model, rapid cooling or *quenching* yields nonminimal results. With sufficiently slow cooling, simulated annealing can provide practical solutions to many optimization problems [6]. It has the further advantage that if certain conditions hold [9], the probability of achieving a global minimum approaches 1.0. This is unlike deterministic heuristic algorithms, in which nonminimal solutions can occur; indeed, our experience [11] is that it is easy to find expressions for which a given heuristic does *not* achieve the minimal solution. The achievement of a global minimum with a probability that approaches 1.0 may not be satisfactory if the probability is low after a reasonably long computation. However, our experience suggests that improved results over all known heuristics are obtained with reasonable computation times for large expressions.

The results of our research are reported in Sections IV and V. Section IV shows the results of simulated annealing on a single expression. To illustrate this algorithm's ability to solve large problems, we choose an expression with 200 minterms and a best known solution of about 90 product terms. We show tradeoffs that can be made between computation time and the optimality of the final result. Section V shows a comparison with other minimization heuristics. The basis is a set of specially selected expressions and a set of randomly generated expressions. Again, we compare the number of product terms and computation time.

II. THE SUM-OF-PRODUCTS MINIMIZATION PROBLEM

An r-valued function, $f(x_1, x_2, \cdots, x_n)$, takes on a value from $\{0,1,...,r-1\}$, for each assignment of values to the variables, which are also r-valued; i.e. $x_i \in \{0,1,...,r-1\}$. r, the radix, is the number of logic values in the system. Because of their widespread use, we choose to represent a function in its sum-of-products form. Because the truncated sum is so easily implemented in multiple-valued PLA's, we choose to use it. A *product term* or *implicant* is expressed as

$$c\ {}^{a_1}x_1^{b_1}\ {}^{a_2}x_2^{b_2}\ \cdots\ {}^{a_n}x_n^{b_n}, \tag{1}$$

where $c \in \{1,2,...,r-1\}$, is a nonzero constant, where the literal function,

$${}^{a_i}x_i^{b_i} = r-1 \text{ if } a_i \le x_i \le b_i ,$$

$$= 0 \qquad \text{otherwise.}$$

and where concatenation is the *min* function; i.e. $x\,y = \min(x, y)$. Since the literal function ${}^{a_i}x_i^{b_i}$ takes on only values 0 and $r-1$, the product (*min*) of literals ((1) without c) is either 0 or $r-1$, while the complete term (with c) takes on values 0 and c. Indeed, (1) is c iff $a_i \le x_i \le b_i$, for *all* i. For two variable functions, it is convenient to represent a product term

as a rectangle of values of c extending from $x_1 = a_1$ to $x_1 = b_1$ and from $x_2 = a_2$ to $x_2 = b_2$.

The sum-of-products expression for any function $f(x_1, x_2, \cdots x_n)$ is some number of product terms summed together using the truncated sum operation, shown as a $+$ on the left-hand side of

$$a + b = \min(r-1, a + b),$$

where the $+$ on the right-hand side is ordinary addition with logic variables viewed as integers. Thus, if this sum should exceed $r-1$, the *min* operation will assign $r-1$ to the logic expression $a + b$.

Any function $f(x_1, x_2, \cdots, x_n)$ can be represented in a sum-of-products form; for example, each assignment of values to the variables that yields a nonzero value for $f(x_1, x_2, \cdots, x_n)$ can be represented as a product term that is 0 for all other assignments of values. Such a product term is called a *minterm*. Summing all minterms, using the truncated sum operation, yields the function $f(x_1, x_2, \cdots, x_n)$.

We view a minimal sum-of-products expression as being any expression with the minimal number of product terms. While the problem of finding one sum-of-products expression for a function is straightforward, as shown in the example immediately above, the problem of finding a *minimal* expression is another matter altogether. The *sum-of-products expression minimization problem* is to find a sum-of-products expression with the fewest product terms for a given function $f(x_1, x_2, \cdots, x_n)$. As stated in the introduction, the only known algorithm for solving this is exhaustive search.

III. SIMULATED ANNEALING

Simulated annealing has been introduced [6] as a means to solve large-scale optimization problems. It is based on a principle in statistical mechanics, in which a low energy crystalline state is achieved by first melting a substance and then slowly cooling it.

A. THE GENERAL PROCESS

A basic computation in simulated annealing is a *move*. A move is a transition from one solution to another solution in the solution space. In the sum-of-products minimization problem, the solution space is the set of all sum-of-product expressions for some given function. Associated with each solution is a cost. In the sum-of-products minimization problem, the cost is the number of product terms. A move can either increase, decrease, and leave the cost unchanged. Intuitively, one favors moves that decrease the cost, since this drives the system to a minimal solution. However, in a solution space with false minima, the exclusive application of cost-decreasing moves can produce nonoptimal solutions. Cost-increasing or *hill-*

climbing moves are also needed if the system is to recover from a local minima.

In simulated annealing, prospective moves are chosen at random. If a move decreases the cost, it is always accepted (taken). If it increases the cost, it is treated probabilistically. That is, a cost-increasing move is accepted with probability $P(\Delta E) = e^{-\Delta E / k_B T}$, where k_B is a constant, the Boltzmann constant (which we choose as 1), T is the temperature, and ΔE is the increase in cost as the result of making the move. When a cost-increasing move is rejected, another prospective move is randomly chosen and the process repeated. Initially, a high temperature is chosen, in which case $P(\Delta E)$ is high. Here, almost all moves are accepted, regardless of whether they increase or decrease the cost. A system that is held in this state for a sufficiently long time is considered to be *melted*. In the melted state, there is no progress toward a minimal solution; rather the system undergoes random changes and is typically far from a minimal solution.

Once the system has persisted in this state, the temperature is reduced and the process repeated. However, the probability of accepting a cost-increasing move is now slightly lower. This process continues, as the decreasing temperature gradually decreases the probability of accepting cost-increasing moves. The result is slow progress toward an optimal state. Eventually, the system reaches a point where there is no further improvement. The system is considered to be *frozen*.

The temperature reduction process is called the *annealing schedule*. It is critical to the attainment of a global minimum. When the temperature is rapidly reduced, a process called *quenching*, the result is often far from optimal. Therefore, a slow decline is preferred, even though this requires more computation time. A typical annealing schedule, and one that we use, is described by

$$T_n = \alpha T_{n-1},$$

where α is between 0.80 and 0.99. Here, the temperature at each stage is a large (but constant) fraction of its former value. Values of α less than 0.80 are considered quenching.

The making of a move in the minimum sum-of-products minimization problem is a two-step process. First, a pair of product terms is randomly chosen. Second, a test is applied to determine if they are equivalent to a single product term. If so, they are replaced by the equivalent product term. Otherwise, they are replaced by a set of two or more product terms. We describe these two steps in the next sections.

B. CHOOSING A PAIR OF PRODUCT TERMS

For two completely separate product terms, there is no prospect of combining them, and such product terms are not considered. The algorithm only considers adjacent product terms. Two product terms are *adjacent* if and only if a minterm of one is either coincident or

adjacent to a minterm of the other. For example, Fig. 1 below shows a function with four pairs of adjacent product terms, 1-2, 3-4, 5-6, and 7-8. Two product terms *combine* if they can be replaced by a single product term. For example, of the four adjacent product terms in Fig. 1, three combine. The pair 3-4 combine to the single product term 3 (3 is said to *absorb* 4), 5-6 combine because these are equivalent to a single product term consisting of a pair of (horizontal) 2's, and 7-8 combine because these are equivalent to a single product term consisting of a pair of (vertical) 2's.

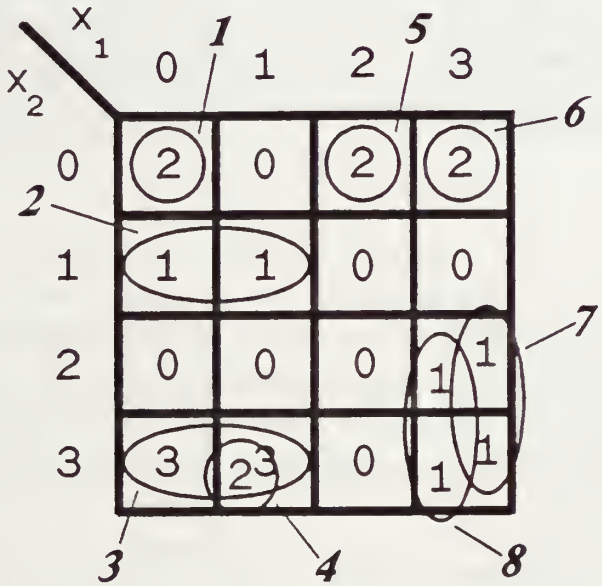


Figure 1. Example of a Function to Be Minimized.

C. OPERATIONS ON A PAIR OF PRODUCT TERMS

We consider two moves, cut-and-combine and reshape. Both choose a pair of product terms, as described above. While, both combine the pair in the same way, each executes the replacement in different ways. Our motivation in investigating two types of moves is the insight gained on how the efficiency of simulated annealing depends on the sophistication of the move.

1. THE CUT-AND-COMBINE MOVE

In the *cut-and-combine* move, the two product terms are combined, if possible, as explained above. However, if not, a cut is performed, as follows. One of the two products is chosen randomly and with equal probability (0.50). If the chosen product term is a minterm of value 1, the move is rejected and another pair of adjacent product terms is chosen. However, if the chosen product term is not a 1 minterm, it is divided. A division can occur along the logic value, in which case, two product terms are formed each with the same literals as

the original product terms, but with coefficients that sum to the coefficient of the original product term. For example, in a 4-valued system, if a product term with coefficient 2 is cut, it is replaced by two product terms each with coefficient 1. If the original product term coefficient is $r-1$, then the coefficients of the divided terms can be anything as long as their truncated sum is $r-1$. A product term divided in such a way is said to undergo a *logical* divide. For example, in a 4-valued system, if a minterm with coefficient 3 is cut, it can be replaced by two minterms each with coefficient 2. A product term can also be divided *geometrically*. In this case, a variable value with a literal range of two or more is chosen and two product terms with the coefficient of the original product terms but divided along that variable are chosen. Of the ways to divide a product term, including all logical and all geometrical divides, one is chosen with a probability that is equivalent to all others (i.e., uniform probability).

2. THE RESHAPE MOVE

The cut-and-combine move is basic. It provides a fundamental cost-increasing move, the cut, where a single product term is converted into two product terms. It also provides a cost-decreasing move, the combine, where two product terms are converted into a single product term. The *reshape* move, like the cut-and-combine, operates on a randomly chosen pair of product terms. Also like the cut-and-combine, it combines the two product terms if a combine is possible. However, for noncombinable product terms, the reshape move proceeds differently.

First, the *consensus* operation is applied. That is, if the two product terms overlap, the consensus of the two product terms is a product term situated at the intersection of the two terms with a coefficient that is the truncated sum of the coefficients of the two product terms. If the two product terms do not overlap, then they must be strictly adjacent. In this case, the consensus is a single product term that is a part of both terms with a coefficient that is the minimum of the coefficient of the two product terms. The part of each product term that contributes to the consensus of the two is the "face" of the intersection that extends along the whole of the variable across which the two product terms are adjacent. Fig. 2 shows an example of the two subcases of the consensus operation. The consensus is indicated by the hatched area. Fig. 2a shows the consensus in the case of overlapping product terms, while Fig. 2b shows the consensus in the case of disjoint product terms. For each of the two product terms, the consensus is subtracted. Of what is left, there are several ways to divide the remaining product terms. Fig. 2 shows one way. From the ways that result in the fewest product terms, one is chosen randomly. Unlike the cut-and-combine move, the reshape move can produce three or more product terms from the original two. Indeed even two different

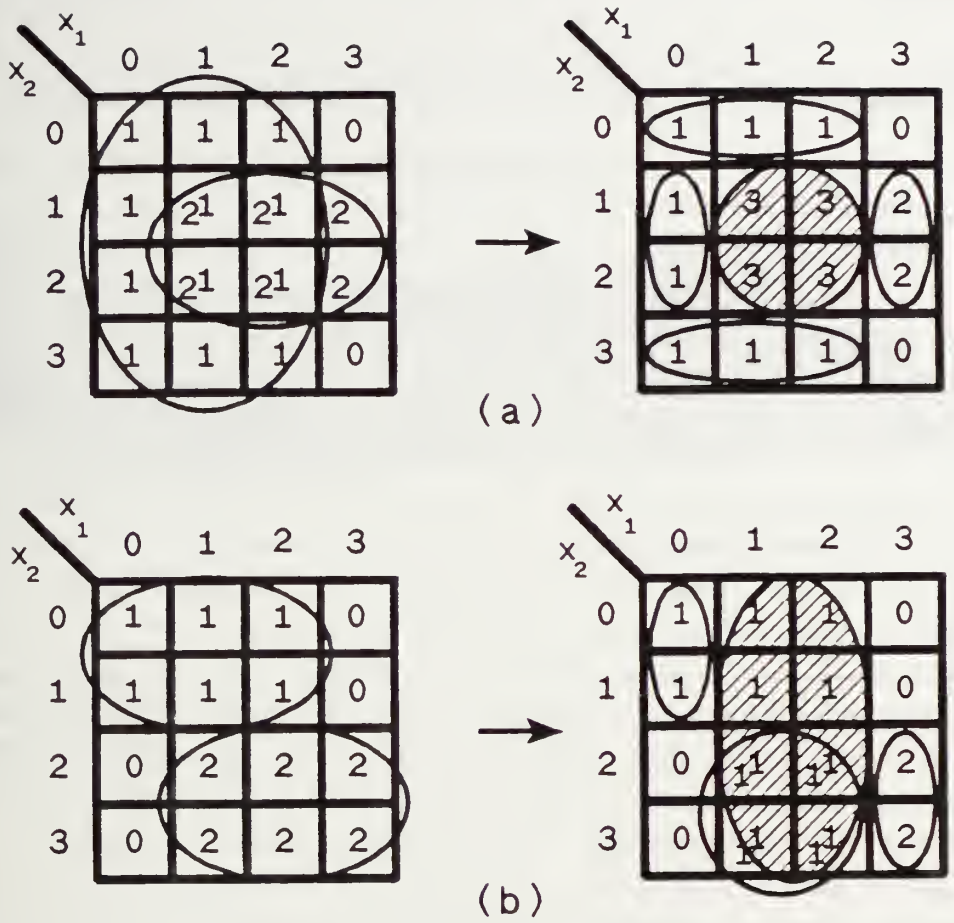


Figure 2. Example of the Consensus Operation.

product terms can result. An example of the latter occurs in the case of product term 2 in Fig. 1 and product term 1 replaced by a 1 minterm. The application of the reshape move yields a product term consisting of a vertical pair of 1's plus a 1 minterm. In this example, the consensus term is the vertical pair of 1's, $1\ {}^{0}x_1^{0}\ {}^{0}x_2^{1}$.

The reshape move suffers from a disadvantage. For example, in a 4-valued system, consider a minimal sum-of-products expression consisting of two product terms with coefficient 2 in the form of a cross. At the intersection, the coefficient is 3. Given an initial solution consisting of five disjoint parts of the cross, there is no path that will allow the reshape move to achieve the minimal solution. That is, the reshape move, while able to form one half of the cross in combination with combines, is unable to form the other half. The best solution is with three implicants. Unlike the cut-and-combine move, the reshape move does not create

product terms that oversum. In this example, as in others, this ability is necessary to achieve a minimal solution.

IV. EXPERIMENTAL RESULTS OF SIMULATED ANNEALING

Unlike previous analyses, which considered expressions with few product terms, we consider, in this section, an expression with significantly more product terms. Fig. 3 shows the result of simulated annealing using the two types of moves, cut-and-combine (a) and reshape (b) applied to a randomly chosen 4-variable 4-valued function with 200 minterms. Prior to the annealing process, this function was minimized using the Dueck and Miller heuristic [3], resulting in a solution of 96 product terms. In both graphs, the number of product terms is plotted horizontally with larger numbers to the right. The temperature is plotted in the axis perpendicular to the page, with higher temperatures in the front. The number of times a visit is made to a solution with some number of product terms is plotted along the vertical axis. Vertical "slices" represent a histogram of the number of times the system is in a solution with the corresponding number of product terms specified along the horizontal axis. Each slice represents one temperature. The slice in the very front represents the highest and starting temperature. It shows how melting takes place. For this temperature, moves transform the initial 96 product term solution into solutions with approximately 275 product terms. The progression goes from left to right, along the front edge of the diagram. Melting occurs quickly; that is, the progression toward solutions with more product terms is seen as a minor vertical deviation until the melted state. This is because, initially, the majority of adjacent pairs of product terms cannot combine. As a result, most moves in the beginning are cost-increasing moves. There is a steady progression to solutions with more product terms, and so solutions with few product terms are visited infrequently. The vertical deviation for such numbers is thus small. However, as the number of solutions increases, more pairs can combine, and, thus, there are more cost-decreasing moves. The mix of cost-increasing and cost-decreasing moves becomes balanced. As a result, visits to states with the same number of product terms become more frequent, and there is a corresponding larger vertical deviation. At the temperature just below the melted state, almost all of the solutions have nearly the same number of product terms, and the vertical deviation is larger than at the initial temperature.

It is interesting that the total number of product terms in the melted state is greater than the number of minterms. In the cut-and-combine move, there are solutions in the melted state that have approximately 275 product terms. This exceeds the 200 nonzero minterms in the initial specification of the expression because of a property of the cut-and-combine move: Given a product term, it is possible to cut it into two product terms identical to the initial product term except that the coefficients of the latter sum to the coefficient of the former.

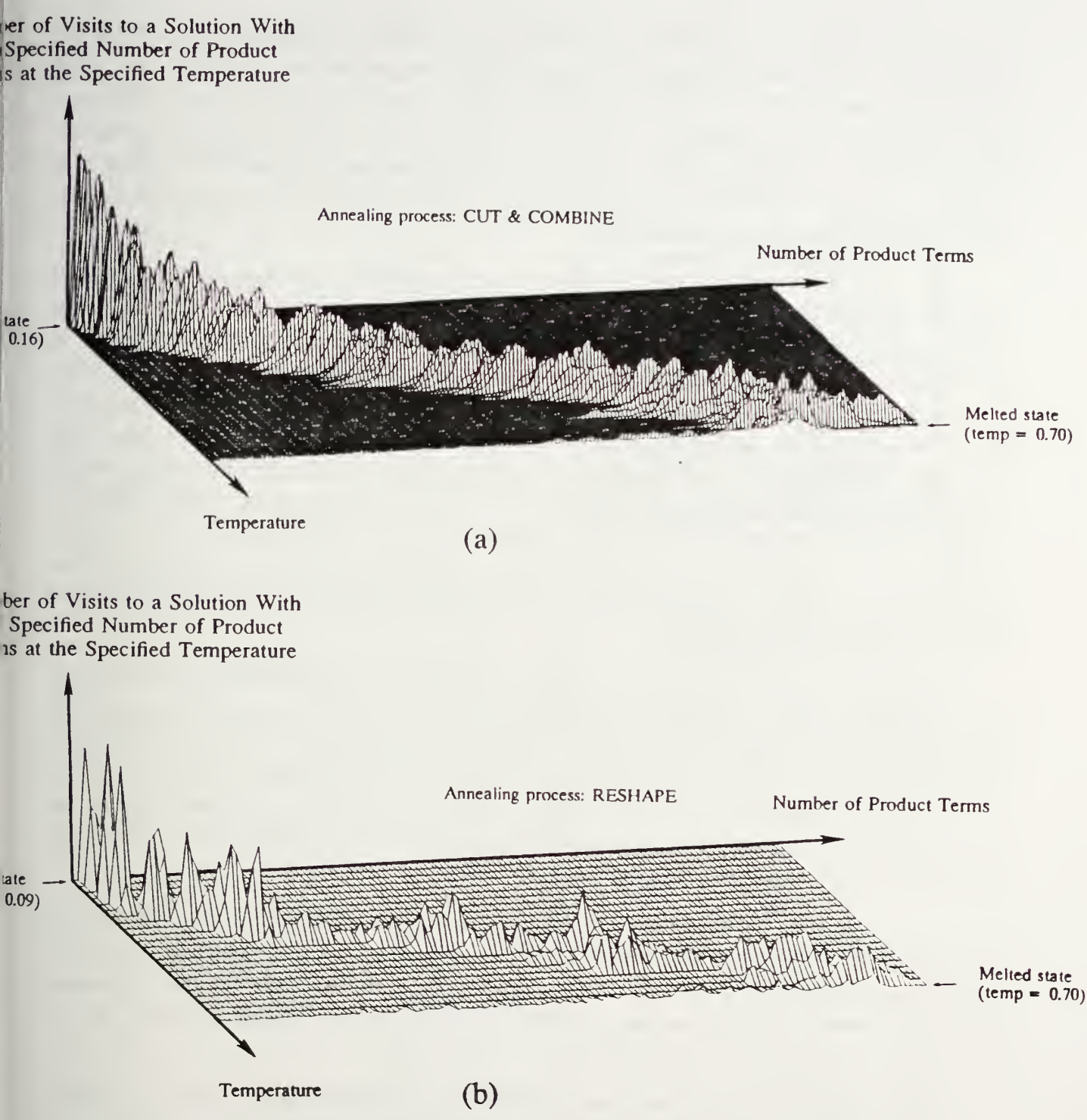


Figure 3. Simulated Annealing Using the (a) Cut-and-combine and the (b) Reshape Moves.

Indeed, if the coefficient of the initial product term is $r-1$, there can be many ways the sum can occur because of oversumming (e.g. when $r = 4$, there are five ways to form the sum of

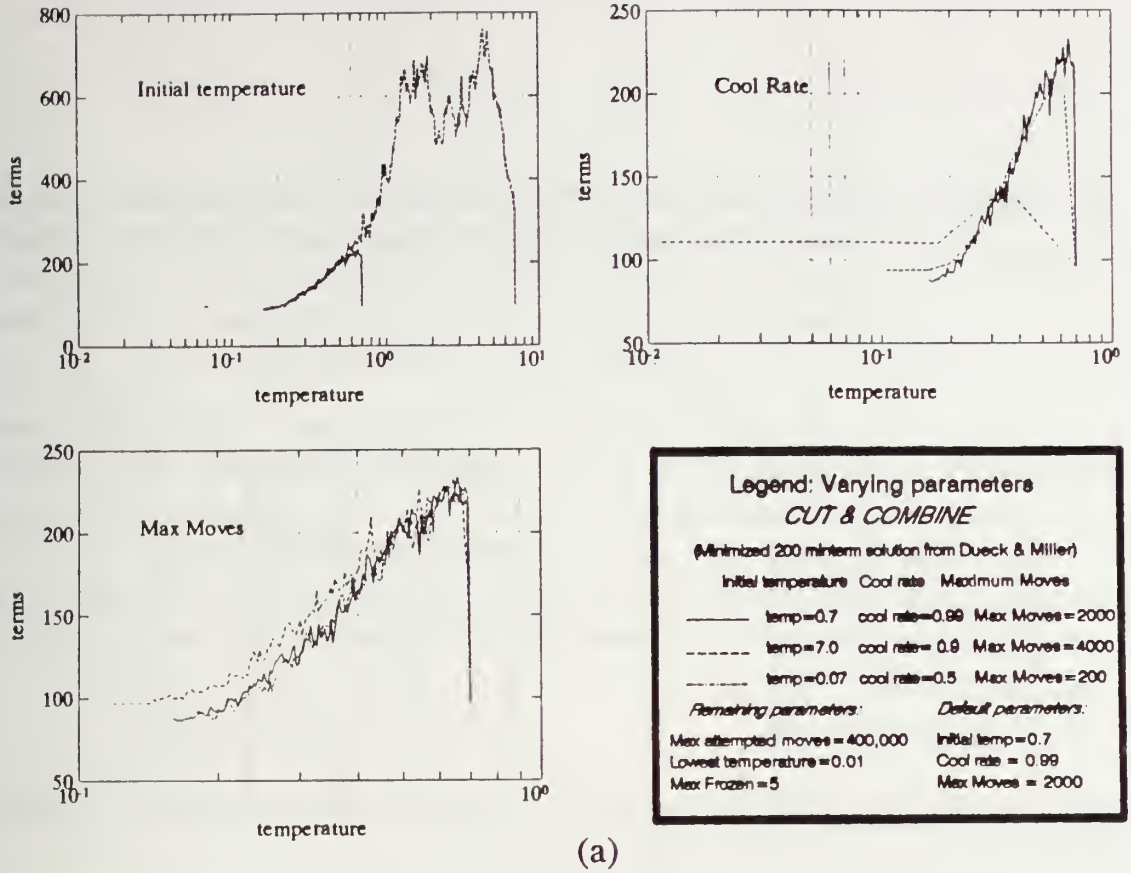
3, versus only one way to form the sum of 2). A similar phenomenon occurs with the reshape move. For example, consider a two-variable 4-valued function consisting two 3's at the opposite corners of a 2 by 3 rectangle with a pair of adjacent 2's in between. This function can yield five product terms by a sequence of reshape moves starting from four minterms. That is, starting with three product terms, two 3 minterms and the pair of 2's, there is a sequence of reshape moves that will produce a five product term solution where each 3 minterm is replaced by a 2 and a 1 minterm.

Once the melted state has been reached, there is a gradual trend toward fewer product terms as the temperature decreases. The temperature axis is logarithmic. That is, every equally spaced temperature slice represents some fraction α of the slice closer to the front. In the case of the cut-and-combine move, α is 0.99, and in the case of the reshape, α is 0.93. The slow migration towards solutions with fewer product terms is evident. As the temperature decreases (moving toward the origin), there is a gradual shift to solutions with fewer product terms, until eventually all transitions are among solutions with very few product terms. In the case of cut-and-combine, a solution of 87 product terms is achieved, while in the case of the reshape, a solution of 84 product terms is achieved. It is interesting that cut-and-combine with a slower rate of temperature decline produced a solution with more product terms than the reshape. The values for α were chosen carefully to provide good solutions with reasonable execution times.

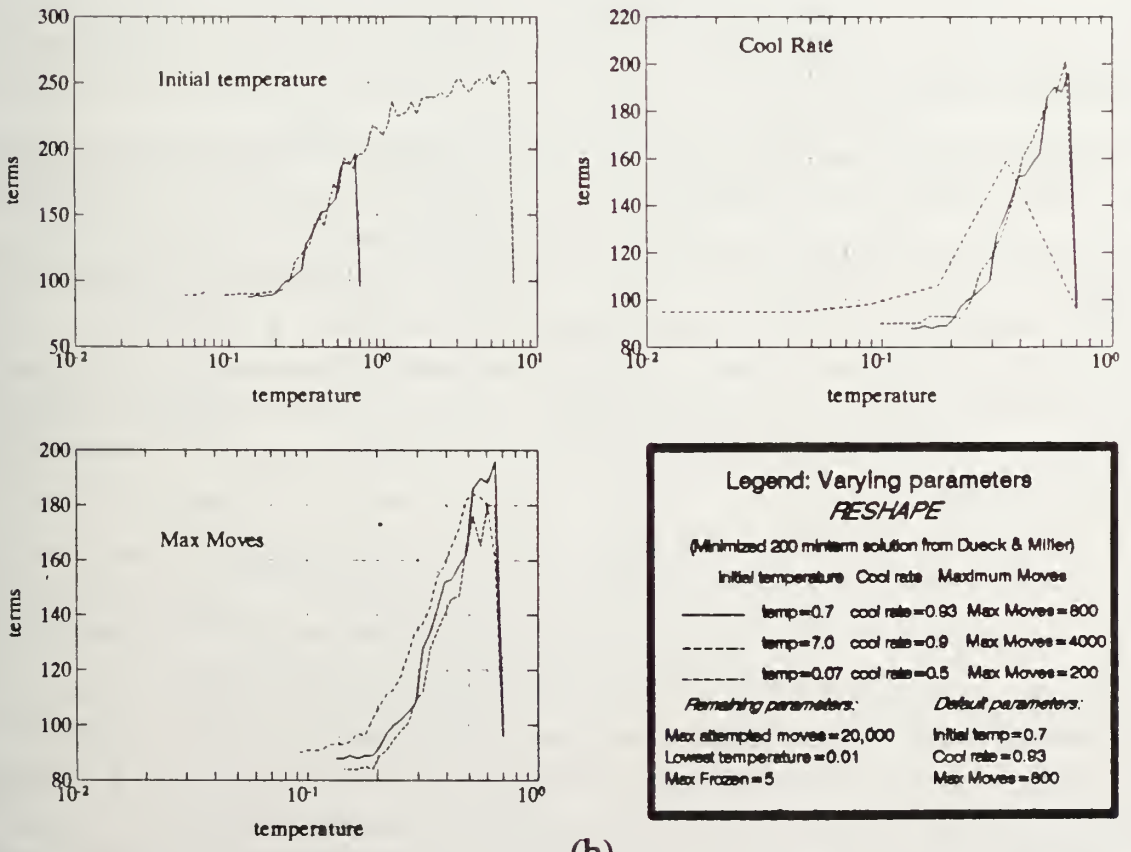
The cut-and-combine requires a total of 91.4 minutes of computation time were required on a Solbourne Series 4 workstation (equivalent to a Sun 110) for the cut-and-combine, while 3.98 minutes were required for the reshape. This illustrates the relation between the annealing schedule and the computation time. With $\alpha = 0.99$, the cut-and-combine exhibits a slower rate of decline in temperature than the reshape move, where $\alpha = 0.93$. Compensating for the large number of temperatures in the cut-and-combine move is the additional time required by the reshape to manipulate the product terms.

A more complete understanding of simulated annealing is gained by plotting the solutions against various parameters. Fig. 4 shows how the solutions depend on different choices for the starting temperature, the cooling rate, and the number of moves made at each temperature. Fig. 4a shows these parameters for the case of the cut-and-combine move, while Fig. 4b shows the parameters for the reshape move.

This data shows an interesting independence on initial temperature. That is, the number of product terms in the final solution tends to be approximately independent on the initial temperature. For each starting temperature, there is a clear period of melting, in which the initial 96 product term solution increases rapidly at the first few temperatures. That is, in these diagrams the heuristic begins at a high temperature and a reasonably small number of product



(a)



(b)

Figure 4. The Number of Product Terms As a Function of Starting Temperature, Cooling Rate, and Number of Moves Made at Each Temperature.

terms. There is a rapid increase in the number of product terms as the temperature decreases, followed by a gradual decrease in the number of product terms going towards the left. This course, as shown in Fig. 4, is relatively unaffected by the starting temperature. This observation applies to large start temperatures. For very low start temperatures, there is a dependence, indeed a degradation in the final solution when the start temperature is low.

A similar statement is not true of the effect of cooling rate. For both the cut-and-combine and reshape move, the quality of the final solution is dependent on the cooling rate. The fastest cooling rate, corresponding to $\alpha = 0.5$, results in solutions that are far from optimal. This is quenching. The plots also show a dependence on the number of moves at each temperature, which is analogous to the time spent at each temperature. Especially, when the maximum number of moves is restricted to a low of 200, there is a clear degradation in the algorithm's performance.

V. COMPARISON OF SIMULATED ANNEALING WITH OTHER MINIMIZATION ALGORITHMS

To achieve a fair comparison of simulated annealing with other heuristic minimization methods, we consider two types of test functions. The first consists of individual functions selected for their unique characteristics. The second consists of functions randomly generated by the HAMLET CAD tool.

In the first set, there are three functions. *Test1* is a randomly chosen 4-valued 3-variable function with 50 minterms. An exhaustive search in HAMLET shows that the exact minimal solution contained 21 product terms. The Dueck and Miller heuristic [3] in HAMLET results in a solution of 24 product terms. This is the form put into the simulated annealing program. After 104.4 minutes of computation time on a Solbourne Series 4 Workstation, cut-and-combine produced a (minimal) solution of 21 product terms, while reshape produced a solution of 21 product terms, but within 2.5 minutes. *Test2* is a 4-valued 4-variable symmetric function with 176 minterms and a minimal solution of 6 product terms. The minimal sum-of-products expression for this is

$$1\,{}^{2}x_1^{3}\,{}^{2}x_2^{3}\,{}^{0}x_3^{3}\,{}^{0}x_4^{3} + 1\,{}^{2}x_1^{3}\,{}^{0}x_2^{3}\,{}^{2}x_3^{3}\,{}^{0}x_4^{3} + 1\,{}^{2}x_1^{3}\,{}^{0}x_2^{3}\,{}^{0}x_3^{3}\,{}^{2}x_4^{3} +$$

$$1\,{}^{0}x_1^{3}\,{}^{2}x_2^{3}\,{}^{2}x_3^{3}\,{}^{0}x_4^{3} + 1\,{}^{0}x_1^{3}\,{}^{2}x_2^{3}\,{}^{0}x_3^{3}\,{}^{2}x_4^{3} + 1\,{}^{0}x_1^{3}\,{}^{0}x_2^{3}\,{}^{2}x_3^{3}\,{}^{2}x_4^{3}.$$

Its special characteristic is that it is difficult to minimize by cut-and-combine. That is, the minimal solution exists among many nonminimal solutions that are easily produced by the random cutting of product terms. The random nature of cut-and-combine makes it difficult to converge to the minimal solution from among the many nonminimal solutions. Reshape, on the other hand, tends to maintain group integrity, and will not introduce miscellaneous logical cuts that tend to move away from the minimal solution. *Test3* is a 4-valued 2-variable

function that was chosen because reshape does not find the minimal solution for it. *Test3* is shown in Fig. 5. This function requires oversumming where the truncated sum indeed

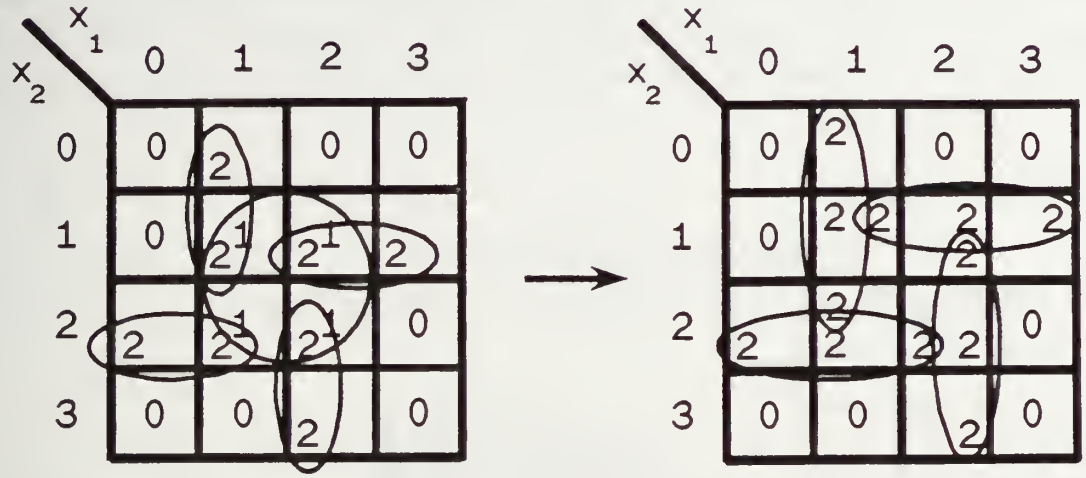


Figure 5. *Test3*, a Test Function.

truncates. Because of this, reshape does not achieve a minimal solution. It is relatively simple, and so cut-and-combine finds the solution easily. Table I shows the results of various algorithms on the three test functions. Our expectation of the relative merits of cut-and-

Heuristic	*Test1*			*Test2*			*Test3*		
	in	**out**	**time**	**in**	**out**	**time**	**in**	**out**	**time**
Cut-&-combine	24	21	6264	14	19	2125.4	5	4	207*
Reshape	24	21	147	14	7	71.0	5	5	4.6
Dueck & Miller	24	24	0.9	14	6	5.7	5	5	0.03
Pomper & Armstrong	24	24	0.4	14	10	3.0	5	5	0.01
Yang & Wang	24	22	8.6	14	10	9.3	5	4	0.12

*This is the total time. The minimal solution was first found in 4.2 secs..

Table I. Number of Product Terms and Computation Time For Three Test Functions As Produced by Five Heuristics.

combine and reshape on *Test2* and *Test3* are borne out. Interestingly, only the Dueck and Miller heuristic found the minimal solution on *Test2*, while cut-and-combine produced a

solution quite far from optimal. For *Test3*, only the cut-and-combine and the heuristic by Yang and Wang [11] achieved the minimal solution.

The second group of tests consists of randomly chosen functions. For this test case, nine ensembles of ten functions each were chosen. Each ensemble consists of 4-valued 4-variable functions with the same number of minterms, a value that ranged from 50 to 250. Fig. 6 shows the results. Cut-and-combine performed best in ensembles having fewer minterms,

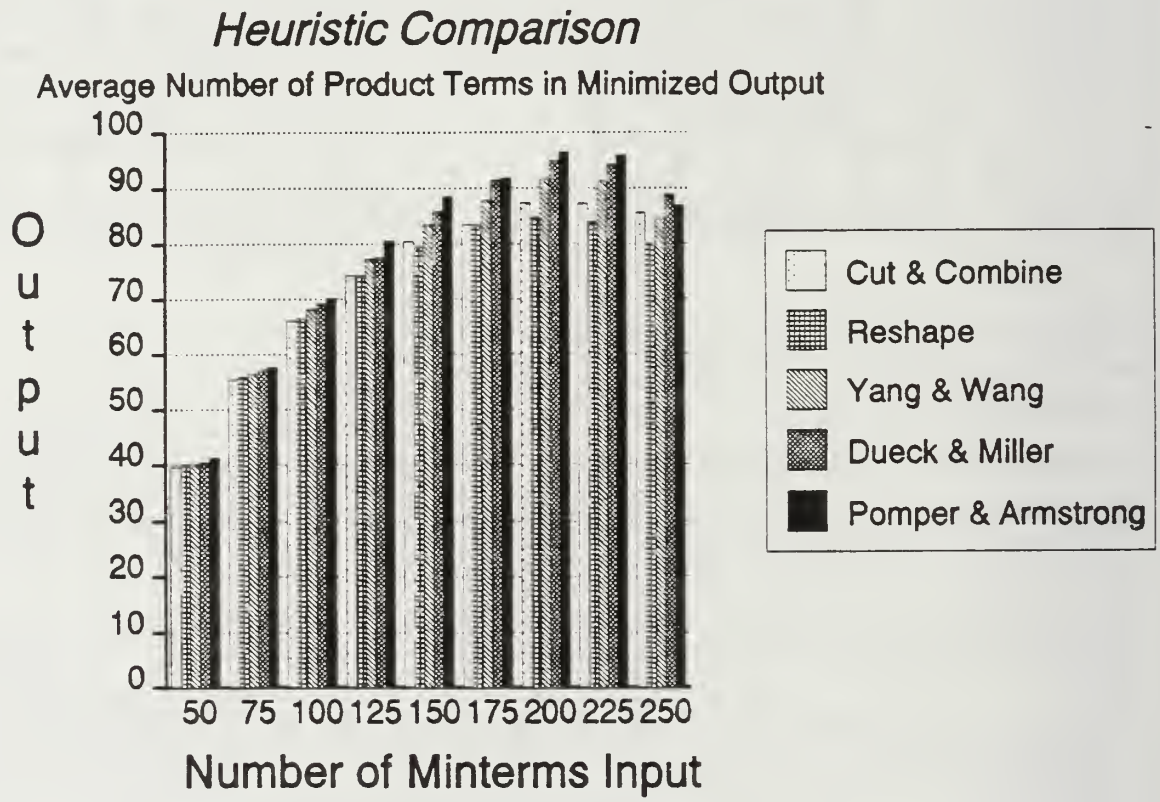


Figure 6. Comparison of Number of Product Terms Produced by Various Heuristics Versus the Number of Minterms Over Random Functions.

while reshape had the best performance on the remaining functions. Fig. 7 shows the execution time of the various heuristics. This shows that the increased "intelligence" exhibited by reshape results in an improved solution, as well as reduced computation time. Both simulated annealing heuristics, on the average, outperformed the other heuristics. This improved

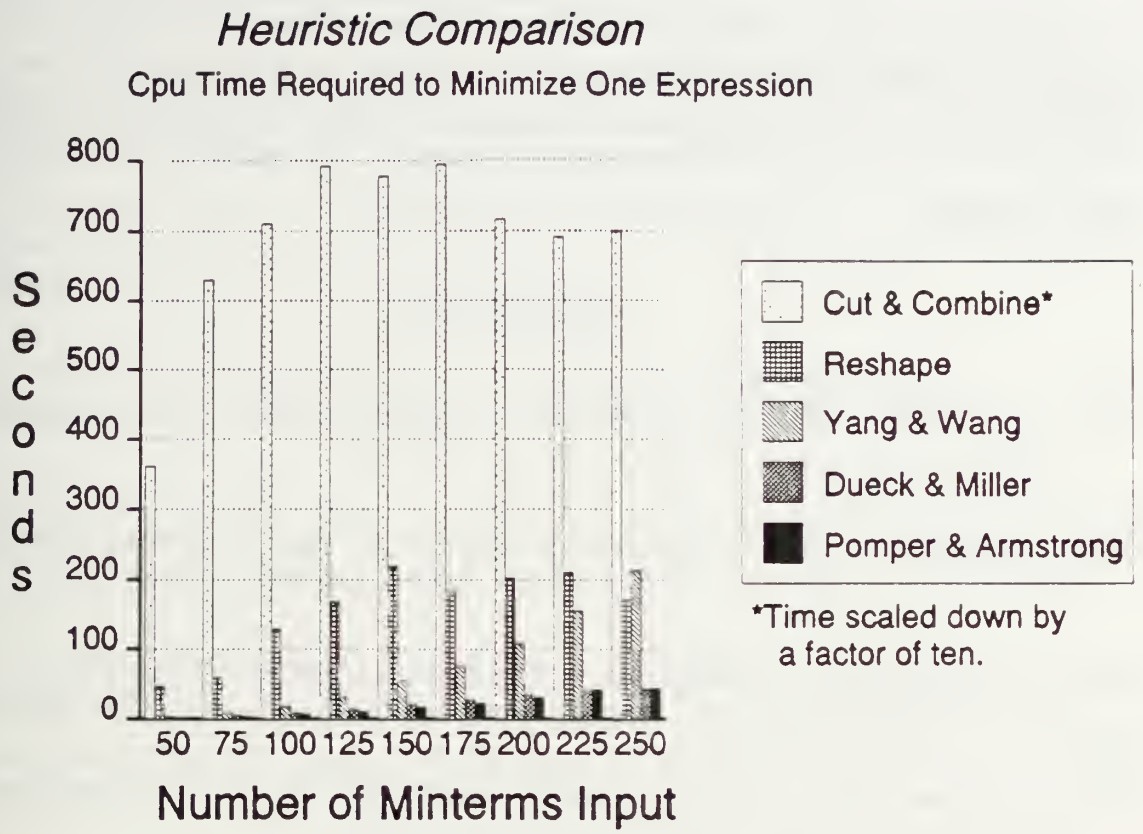


Figure 7. Comparison of Execution Time Required by the Various Heuristics Versus the Number of Minterms Over Random Functions.

performance is not without a price. Computation times are higher.

VI. CONCLUDING REMARKS

This investigation of the use of simulated annealing in finding minimal sum-of-products expressions has been encouraging. First, the time of computation is easily controlled; one can choose a slow annealing schedule, and, in so doing, achieve a solution that tends to be closer to optimum, or a fast schedule with less likelihood of achieving the optimum. Second, simulated annealing has general applicability, and there is the prospect of applying it to further problems in multiple-valued logic circuit design, e.g. layout and routing. Indeed, it may be represent the means to go on to more complex structures than PLA's, thereby achieving even

more compact circuits.

We have shown two algorithms, the simple cut-and-combine and the reshape. The latter requires more computation time doing an individual move, but yields good solutions with less computation time overall than the cut-and-combine. However, for expressions representing few minterms, the cut-and-combine move is superior. Both represent improvements to all known heuristics.

REFERENCES

[1] P. W. Besslich, "Heuristic minimization of MVL functions: A direct cover approach," *IEEE Trans. on Comput.*, February 1986, pp. 134-144.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic, Boston 1984.

[3] G. W. Dueck and D. M. Miller, "A direct cover MVL minimization using the truncated sum," *Proc. of the 17th Inter. Symp. on Multiple-Valued Logic*, May 1987, pp. 221-227.

[4] D. L. Isaacson and R. W. Madsen, *Markov chains: theory and applications*, John Wiley & Sons, New York, London, Sydney, and Toronto, 1976.

[5] H. G. Kerkhoff and J. T. Butler, "Design of a high-radix programmable logic array using profiled peristaltic charge-coupled devices," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 100-103.

[6] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, No. 4598, 13 May 1983, pp. 671-680.

[7] G. Pomper and J. A. Armstrong, "Representation of multivalued functions using the direct cover method," *IEEE Trans. on Comput.* Sept. 1981, pp. 674-679.

[8] D. L. Ostapko, R. G. Cain, and S. J. Hong, "A practical approach to two-level minimization of multiple-valued logic," *Proc. of the Inter. Symp. on Multiple-Valued Logic*, May 1974, pp. 168-182.

[9] F. Romeo and A. Sangiovanni-Vincentelli, "Probabilistic hill climbing algorithms: properties and applications," ERL, College of Engineering, University of California-Berkeley, CA, March 1984.

[10] T. Sasao, "On the optimal design of multiple-valued PLA's," *Proc. of the Inter. Symp. on Multiple-Valued Logic* May 1986 pp. 214-223.

[11] P. P. Tirumalai and J. T. Butler, "Minimization algorithms for multiple-valued programmable logic arrays", *IEEE Transactions on Computers*, Feb. 1991, pp. 167-177.

[12] P. P. Tirumalai and V. G. Vadakkencherry, "Parallel algorithms for minimizing multiple-valued programmable arrays, *Proceedings of the 21st International Symposium on Multiple-Valued Logic*, May 1991, pp. 287-295.

[13] C. Yang and O. Oral, "Experiences of parallel processing with direct cover algorithms for multiple-valued logic minimization," preprint.

[14] C. Yang and Y.-M. Wang, "A neighborhood decoupling algorithm for truncated sum minimization," *Proceedings of the 20th International Symposium on Multiple-Valued Logic*, May 1990, pp. 153-160.

[15] J. Yurchak and J. T. Butler, "HAMLET - An expression compiler/optimizer for the implementation of heuristics to minimize multiple-valued programmable logic arrays", *Proceedings of the 20th International Symposium on Multiple-Valued Logic*, May 1990, pp. 144-152.

APPENDIX A: FORMAL SPECIFICATION OF THE PROGRAM TO SOLVE THE MINIMAL SUM-OF-PRODUCTS PROBLEM BY SIMULATED ANNEALING

The following is a formal description of the application of simulated annealing to the minimization of a multiple-valued sum-of-products expression. The input is a set of product terms, ProductTermSet, that sum to the given function. The output is a minimized set of product terms that is returned as ProductTermSet.

algorithm SimulatedAnnealing(ProductTermSet);

/* Minimize a multiple-valued expression, ProductTermSet, by simulated annealing with the following global constraints. Default values are shown in parentheses. All such values can be overridden by the user. Typical ranges are indicated by brackets.

InitialTemp - T_0, temperature at which simulated annealing begins (0.7) [0.5 - 10.0].

LowestTemp - The lowest temperature below which simulated annealing is no longer applied (0.01).

MoveType - Type of move taken at each attempted move (Cut-and-Combine) [Cut-and-Combine or Reshape].

MaxMoves - Maximum allowed number of moves completed at each temperature (Cut-and-Combine: 13 × number of minterms. Reshape: 4 × number of minterms.).

MaxAttemptedMoves - Largest number of attempted moves made before the current temperature is abandoned (Cut-and-Combine: 210 × MaxMoves. Reshape: 25 × MaxMoves.).

MaxFrozen - Longest sequence of temperatures at which the number of attempted moves is MaxAttemptedMoves (4) [2-10]. At this temperature, many attempted moves are needed to achieve completed moves, and the solution is considered to be frozen. The simulated annealing is stopped, since little progress is achieved with continued computational effort.

CoolRate - α, the factor used to determine the next temperature, i.e. $T_n = \alpha T_{n-1}$ (Cut-and-Combine: 0.99. Reshape: 0.93.). [0.50 - 0.99].

*/

BestProductTermSet ← ProductTermSet;
CurrTemp ← InitialTemp;
Frozen ← 0;
while (Frozen ≤ MaxFrozen) **and** (CurrTemp ≥ LowestTemp) **do**;
    begin
        AttemptedMoves ← 0;

```
        Moves ← 0;
        while (Moves ≤ MaxMoves) and (AttemptedMoves ≤ MaxAttemptedMoves)
                                  and (there are adjacent pairs in ProductTermSet) do;
            begin
                ChooseAdjacentProductTerms(ProductTermSet, PT1, PT2);
                AttemptaMove(PT1, PT2, CurrTemp, MoveType, PT1plusPT2, MoveMade);
                AttemptedMoves ← AttemptedMoves + 1;
                if (MoveMade) then
                    Moves ← Moves + 1;
                    ProductTermSet ← ProductTermSet - PT1 - PT2 ∪ PT1plusPT2;
                    if ( |ProductTermSet| < |BestProductTermSet| ) then
                        BestProductTermSet ← ProductTermSet;
                    end;
                end;
            end;
        if (AttemptedMoves > MaxAttemptedMoves) then Frozen ← Frozen + 1;
        else Frozen ← 0;
        CurrTemp ← CoolRate × CurrTemp;
        end;
    end;
    ProductTermSet ← BestProductTermSet;
stop;


procedure AttemptaMove(PT1, PT2, CurrTemp, MoveType, PT1plusPT2, MoveMade);

  /*  Attempt a move at the CurrTemp
        PT1plusPT2 - Set of product terms equivalent to PT1 + PT2 (If a combine is
            possible, for example, PT1plusPT2 contains the single equivalent product
            term).
        MoveMade - Indicates whether or not an attempted move was completed (true or
            false).
  */

  MoveMade ← false;
  if (PT1 and PT2 combine) then
      Combine(PT1, PT2, PT1plusPT2);
      MoveMade ← true;
  else if (MoveType = Reshape) then
      Reshape(CurrTemp, PT1, PT2, PT1plusPT2, MoveMade);
  else if (MoveType = Cut-and-Combine) then
```

```
        Cut(CurrTemp, PT1, PT2, PT1plusPT2, MoveMade);
    end;
return;


procedure Combine(PT1, PT2, PT1plusPT2);

    /*  Combine PT1 and PT2 into a single product term.
    */

        if (PT2 absorbs PT1) then
            PT1plusPT2 ← PT2;
        else if (PT1 absorbs PT2) then
            PT1plusPT2 ← PT1;
        else if (PT1 and PT2 overlap) then
            PT1plusPT2 is assigned the same literal structure as PT1 and PT2;
            Coef(PT1plusPT2) ← Coef(PT1) + Coef(PT2);
        else if (PT1 and PT2 are disjoint) then
            PT1plusPT2 is assigned the same literal structure as PT1 and PT2 except for the
              variable over which they are disjoint;
            literal(PT1plusPT2) ← literal(PT1) ∪ literal(PT2), where the literal coincides
              with the variable over which PT1 and PT2 are disjoint;
        end;
return;


procedure Reshape(CurrTemp, PT1, PT2, PT1plusPT2, MoveMade);

      /*  Attempt to reshape the product terms.
              Random() - Produces a random number between 0.0 and 1.0.
              RandomSharp(PT, PTConsensus, P) - A routine that accepts two product
                  terms, PT and PTConsensus, where Coef(PT) ≥ Coef(PTConsensus)
                  returns a set of product terms, P.  P consists of a minimal set of product
                  terms that covers all minterms in PT ∩ PTConsensus each having a
                  coefficient of value, Coef(PT) - Coef(PTConsensus), (if that value is
                  greater than zero) and all minterms in PT but not in PTConsensus each
                  having a coefficient of value Coef(PT).  From all choices of a minimal
                  set, one is chosen randomly and with uniform probability.
      */

      MoveMade ← false;
      if (Random() ≥ e^(-ReshapeCost(PT1,PT2)/CurrTemp)) then return;
```

else if (PT1 overlaps PT2) **then**

PTConsensus ← A product term consisting of minterms common to PT1 and PT2 with a coefficient that is the (truncated) sum of the coefficients of PT1 and PT2;

else if (PT1 and PT2 are adjacent but have no common minterms) **then**

PTConsensus ← A product term consisting of minterms included in the face of the adjacency and extending in the variable perpendicular to the adjacency with a coefficient that is the minimum of the coefficients of PT1 and PT2;

end;

RandomSharp(PT1, PTConsensus, P1);

RandomSharp(PT2, PTConsensus, P2);

PT1plusPT2 ← PTConsensus $\cup$ P1 $\cup$ P2;

MoveMade ← true;

return;

procedure Cut(CurrTemp, PT1, PT2, PT1plusPT2, MoveMade);

/* Attempt to divide one of the product terms. */

MoveMade ← false;

if (Random() $\geq e^{-1/CurrTemp}$) **then return**;

else if (Random() < 0.5) **then**

RandomDivide(PT1, PTDivided, MoveMade);

PT1plusPT2 = PTDivided $\cup$ PT2;

else RandomDivide(PT2, PTDivided, MoveMade);

PT1plusPT2 = PT1 $\cup$ PTDivided;

end;

return;

procedure RandomDivide(PT, PTDivided, MoveMade);

/* A function that accepts a product term, PT, and creates two product terms, PTA and PTB, such that PTDivided = {PTA, PTB}. Let

$$PT = c\ {}^{a_1}x_1^{b_1}\ {}^{a_2}x_2^{b_2}\ \cdots\ {}^{a_n}x_n^{b_n},$$

where $c \in \{1, 2, ..., r-1\}$, is a nonzero constant and $a_i \leq b_i$.

This routine computes the number of ways a product term can be cut and chooses one randomly with uniform probability. It returns with MoveMade set to true.

When PT is a minterm with coefficient 1, there is no way to cut it, and a return is executed with MoveMade set to false.

*/

MoveMade ← false;

if (c < r - 1) **then** LogicalCuts ← $\left\lfloor \frac{c}{2} \right\rfloor$;

else if (c = r - 1) **then** LogicalCuts ← $\left\lceil \frac{r}{2} \right\rceil \left\lceil \frac{r+1}{2} \right\rceil - 1$;

end;

GeometricalCuts ← $\sum_{i=1}^{n} a_i - b_i$;

TotalCuts ← LogicalCuts + GeometricalCuts;

if (TotalCuts = 0) **then return**;

else if (Random() < LogicalCuts/TotalCuts) **then**

Perform the corresponding logical cut, creating PTA and PTB;

else

Perform the corresponding geometrical cut, creating PTA and PTB;

end;

PTDivided ← {PTA, PTB};

MoveMade ← true;

return;

APPENDIX B: VERIFICATION OF THE PROGRAM

Because the simulated annealing program is probabilistic, it is difficult to verify its correctness. However, for small problems, we can derive the probability that the program will achieve various solutions. Then, if the program is run for a sufficiently long time, we can derive experimental values for the expected values of the probability that the program has achieved a specific solution. Comparing calculated values with experimental values can provide a sense for the program's correctness.

Simulated annealing is a Markov chain, where states in the chain correspond to some configuration of product terms, i.e. a solution. As an example, consider the two product terms, 1 and 2, in Fig. 1. Viewing these as a single expression, there are only six product terms that cover this. However, there are only five ways these can be combined to form the function. These are shown as five circles in Fig. 8 below. For example, the circle shown on the left, State 1, corresponds to the unique minimal sum-of-products expression for this func-

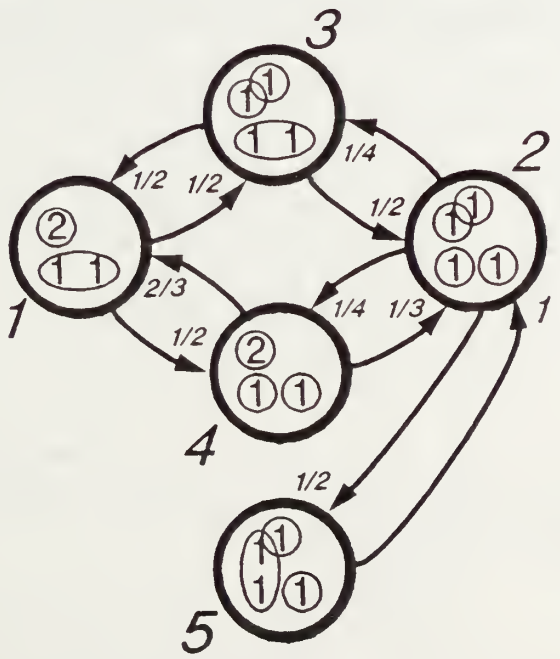


Figure 8. State Transitions in the Markov Chain Model of Simulated Annealing.

tion. It has two product terms. The circle on the right corresponds to the sum-of-products expression with the largest number of product terms. It has four product terms. Arcs between states correspond to transition probabilities in the melted state, when cost-increasing moves are accepted with probability 1.0. For example, in State 3, there are two possible moves, one to State 1 and one to State 2. The configuration of product terms corresponding to State 1 consists of two single 1 product terms which sum to form the 2 of the function and a pair of adjacent 1's. There are three pairs of product terms, all of which are chosen with

probability $\frac{1}{3}$. If the pair of single 1's are chosen, they combine into a single 2, and there is a transition to State 1. If either of the other two pairs are chosen, a combination is not possible and one of the implicants involved is chosen with probability 0.5. If the chosen implicant is the pair of 1's (in each case), it is divided, and there is a transition to State 2. However, if the single 1 is chosen, then the choice of pairs is repeated. The probability that the move is to State 1 is $\frac{\frac{1}{3}}{\frac{1}{3}+2\frac{1}{3}\frac{1}{2}}=\frac{1}{2}$, while the probability that the move is to State 2 is $\frac{2\frac{1}{3}\frac{1}{2}}{\frac{1}{2}+2\frac{1}{3}\frac{1}{2}}=\frac{1}{2}$. These probabilities are shown as weights to the arcs from State 3. The probabilities of other state transitions are shown.

It is convenient to represent the state transition diagram of Fig. 8 as a matrix, the *transition matrix*. This is a complete representation of the probabilities of transition among the various states. Specifically, element p_{ij} is the probability of going from State i to State j. Because the transition probabilities from each state must sum to 1.0, $\sum_j p_{ij} = 1.0$. For the state transitions shown in Fig. 8, we have the transition matrix S, as follows:

$$S = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

If $\boldsymbol{P} = [p_0, p_1, \cdots, p_n]$ is a row matrix, where p_i is the probability of State i initially, then $\boldsymbol{P}\,S$ represents the probabilities after one transition. Similarly, $\boldsymbol{P}\,S^k$ represents the probabilities after k transitions. For example, S^{20} is

$$S^{20} = \begin{bmatrix} 0.412 & 0.412 & 0 & 0 & 0 \\ 0.588 & 0.588 & 0 & 0 & 0 \\ 0 & 0 & 0.353 & 0.353 & 0.353 \\ 0 & 0 & 0.353 & 0.353 & 0.353 \\ 0 & 0 & 0.294 & 0.294 & 0.294 \end{bmatrix}.$$

The form of this matrix shows a consistency in the final state. For example, if the initial state is either 1 or 2 (either $p_1 = 1.0$ or $p_2 = 1.0$), after 20 transitions, the probability of State 1 or 2 is 0.412 or 0.588, respectively. If the initial state is 3, 4, or 5 (either $p_3 = 1.0$, $p_4 = 1.0$ or $p_4 = 1.0$), then the probability of States 3, 4, or 5 is 0.353, 0.353, or 0.294, respectively. This independence of start state is common in Markov chains. We can apply a formal analysis. By applying a standard matrix transformation [4, pp. 127-132], S can be expressed as

$$S = L^{-1} \Lambda L ,$$

where the rows of L are the left eigenvectors $x_1, x_2, \cdots, x_m$ of S, corresponding to eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_m$, respectively, and Λ is a diagonal matrix with entries $\lambda_1, \lambda_2, \cdots, \lambda_m$. Further, S^k can be expressed as

$$S^k = L^{-1} \Lambda L = \sum_{i=1}^{m} \lambda_i^k y'_i x_i ,$$

where x_i is the i-th row of L and y'_i is the i-th column of L^{-1}. Using MACSYMA, an algebraic and symbolic manipulation package, we find the eigenvalues for S to be $\lambda_1 = -\frac{\sqrt{42}}{12}$, $\lambda_2 = \frac{\sqrt{42}}{12}$, $\lambda_3 = -1$, $\lambda_4 = 1$, and $\lambda_5 = 0$. Let $B \sim A^k$ mean $\lim_{n \to \infty} b/a = 1$, for every element b in square matrix B and for the corresponding element a in square matrix A^k. Then, we can write

$$S^k \sim (-1)^k y'_3 x_3 + y'_4 x_4 .$$

y'_3, x_3, y'_4, and x_4 can be calculated from the eigenvectors of S, as produced by MACSYMA, and we have, for even k.

$$S^k \sim \begin{bmatrix} \frac{14}{34} & \frac{14}{34} & 0 & 0 & 0 \\ \frac{14}{34} & \frac{14}{34} & 0 & 0 & 0 \\ 0 & 0 & \frac{12}{34} & \frac{12}{34} & \frac{10}{34} \\ 0 & 0 & \frac{12}{34} & \frac{12}{34} & \frac{10}{34} \\ 0 & 0 & \frac{12}{34} & \frac{12}{34} & \frac{10}{34} \end{bmatrix} ,$$

while for odd k,

$$S^k \sim \begin{bmatrix} 0 & 0 & \frac{12}{34} & \frac{12}{34} & \frac{10}{34} \\ 0 & 0 & \frac{12}{34} & \frac{12}{34} & \frac{10}{34} \\ \frac{14}{34} & \frac{14}{34} & 0 & 0 & 0 \\ \frac{14}{34} & \frac{14}{34} & 0 & 0 & 0 \\ \frac{14}{34} & \frac{14}{34} & 0 & 0 & 0 \end{bmatrix} .$$

Recognizing that for an arbitrary k, the probability that it is even or odd is 0.5, we can obtain the probabilities of specific states as follows:

State	Probability of the State
1	0.206
2	0.294
3	0.176
4	0.176
5	0.147

Table II. Probability of Various States After Many Transitions.

This shows that, in the melted state, where moves that increase the number of product terms are all accepted, the probability of the minimal state, State 1, after many transitions is 20.6%. The most likely state, however, corresponds to the largest number of product terms. It is interesting that the "false minimum" state, State 5, has a lower probability than the true minimum.

This analysis corresponds to the highest temperature, where all moves are accepted. At lower temperatures, moves that increase the cost by ΔE are accepted with probability $P(\Delta E) = e^{-\Delta E / k_B T}$. In our case, there is only one positive value of ΔE, 1, and we denote the probability of accepting such a move as p. The calculation of the probabilities of transition from the various states is straightforward. The corresponding transition matrix is

$$S_p = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{\frac{1}{2}}{\frac{1}{2}+\frac{1}{2}p} & \frac{\frac{1}{2}}{\frac{1}{2}+\frac{1}{2}p} & 0 & 0 & 0 \\ \frac{\frac{2}{3}}{\frac{2}{3}+\frac{1}{3}p} & \frac{\frac{1}{3}}{\frac{2}{3}+\frac{1}{3}p} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

In an analysis similar to that performed above on the melted state, the probabilities that the system is in a specific state, can be calculated to determine their dependence on p (by computing S_p^{20}). The following table shows the results. It can be seen that, as p decreases, the probability of State 1 approaches 0.5, while the probability of State 5 decreases to 0.0. Because a transition is required after each state, the probability of any one state cannot exceed 0.5. Thus, the probability of State 1 approaches the maximum value as p approaches 0.0. This also shows that the probability of the false minimum state, State 5, approaches 0.0 as p

decreases to 0.0.

p	Probability of State				
	1	2	3	4	5
1.0	0.206(0.206)	0.294(0.294)	0.176(0.175)	0.176(0.176)	0.147(0.148)
0.5	0.289(0.294)	0.211(0.206)	0.197(0.198)	0.197(0.200)	0.105(0.102)
0.25	0.365(0.364)	0.135(0.136)	0.217(0.217)	0.217(0.215)	0.067(0.068)
0.125	0.422(0.421)	0.078(0.079)	0.230(0.228)	0.230(0.232)	0.039(0.040)
0.0625	0.457(0.456)	0.043(0.044)	0.239(0.241)	0.239(0.237)	0.021(0.022)
0.03125	0.478(0.477)	0.022(0.023)	0.244(0.245)	0.244(0.244)	0.012(0.011)
0.015625	0.489(0.489)	0.011(0.011)	0.247(0.245)	0.247(0.249)	0.005(0.005)
0.0078125	0.494(0.495)	0.006(0.004)	0.249(0.248)	0.249(0.250)	0.003(0.001)

Table III. Probability of Various States After Many Transitions
as a Function of p, the Probability of Accepting a Cost-Increasing Move.

Shown in parenthesis are experimentally derived values for the probabilities of the various states. These were obtained by running the simulated annealing with the temperatures shown for 100,000 moves. As can be seen, the experimental values match closely the derived values. That is, in the worst case, the difference between the experimental value and the calculated value is 0.5%, for State 2 with $p = 0.5$.

INITIAL DISTRIBUTION LIST

1.	Dr. George Abraham, Code 1005 Office of Research and Technology Naval Research Laboratories 4555 Overlook Ave., N.W. Washington, DC 20375	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Research Administration, Code 012 Naval Postgraduate School Monterey, CA 93943	1
4.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
5.	Prof. Jon T. Butler Department of Electrical and Computer Eng. Naval Postgraduate School Monterey, CA 93943-5004	2
6.	LCDR John M. Yurchak R.D. #6, P.O. Box 268 Muncy, PA 17756a	2
7.	Dr. Robert Williams Naval Air Development Center, Code 5005 Warminster, PA 18974-5000	1
8.	Prof. Michael A. Morgan, Chairman Department of Electrical and Computer Eng. Naval Postgraduate School Monterey, CA 93943-5004	1
9.	Dr. James Gault U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709	1

19. Dr. Clifford Lau — 1
Office of Naval Research
1030 E. Green Str.
Pasadena, CA 91106-2485

20. Dr. John Neff — 1
DARPA/DSO
1400 Wilson Blvd.
Arlington, VA 22209

21. Dr. William Miceli — 1
Office of Naval Research
495 Summer St.
Boston, MA 02210-2109

22. Dr. Harold Szu — 1
Naval Research Laboratories
Code 5750
4555 Overlook Ave., N.W.
Washington, DC 20375

23. Dr. Matthew Kabrisky — 1
Department of Electrical Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

24. Dr. James Suttle — 1
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709